intel *Development Solutions*

ASM86
Pocket Reference
for DOS Systems

Order Number: 122387-001

CONTENTS

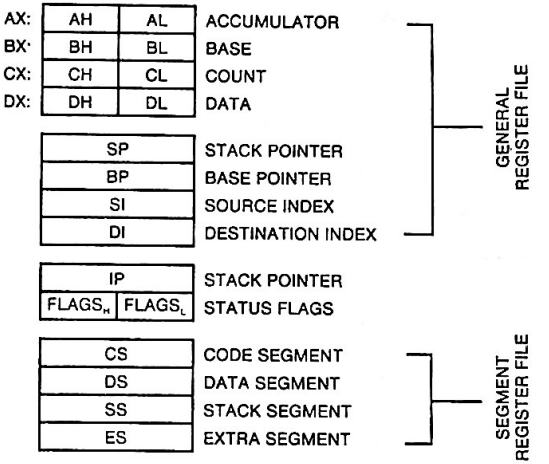
Notational Conventions

color — Indicates user input for interactive examples.

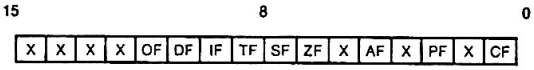
[ ] — Brackets indicate optional arguments or parameters.

<cr> — Indicates a carriage return.

8086 Register Model



Instructions that reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:



X = Don't Care

Flags

AF: AUXILIARY CARRY — BCD
CF: CARRY FLAG
DF: DIRECTION FLAG (STRINGS)
IF: INTERRUPT ENABLE FLAG
OF: OVERFLOW FLAG (CF SF)
PF: PARITY FLAG
SF: SIGN FLAG
TF: TRAP (SINGLE STEP FLAG)
ZF: ZERO FLAG

Operand Summary

"reg" field Bit Assignments:

Word Operand	Byte Operand	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Second Instruction Byte Summary

mod xxx r/m

mod	Displacement
00	DISP = 0*, disp-low and disp-high are absent
01	DISP = disp-low sign-extended to 16-bits, disp-high is absent
10	DISP = disp-high: disp-low
11	r/m is treated as a "reg" field

r/m	Operand Address
000	(BX) + (SI) + DISP
001	(BX) + (DI) + DISP
010	(BP) + (SI) + DISP
011	(BP) + (DI) + DISP
100	(SI) + DISP
101	(DI) + DISP
110	(BP) + DISP*
111	(BX) + DISP

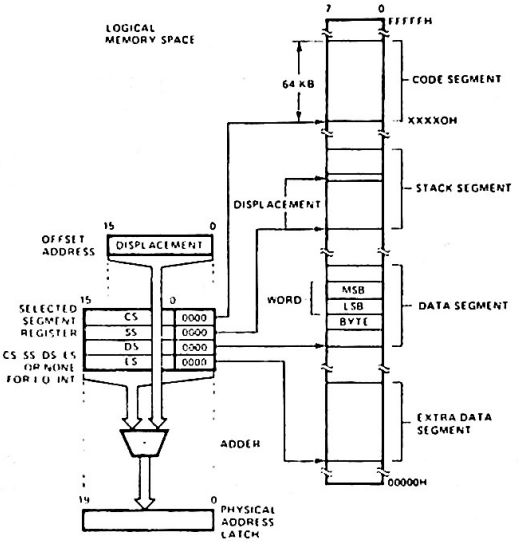
DISP follows 2nd byte of instruction (before data if required).

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

Operand Address (EA) Timing (Clocks):

Add 4 clocks for word operands at ODD ADDRESSES.
Immed Offset = 6
Base (BX, BP, SI, DI) = 5
Base + DISP = 9
Base + Index (BP + DI, BX + SI) = 7
Base + Index (BP + SI, BX + DI) = 8
Base + Index (BP + DI, BX + SI) + DISP = 11
Base + Index (BP + SI, BX + DI) + DISP = 12

Memory Segmentation Model



Segment Override Prefix

0 0 1 reg 1 1 0

Timing: 2 clocks

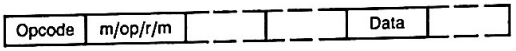
Use of Segment Override

Operand Register	Default	With Override Prefix
IP (code address)	CS	Never
SP (stack address)	SS	Never
BP (stack address or stack marker)	SS	BP + DS or ES, or CS
SI or DI (not incl. strings)	DS	ES, SS, or CS
SI (implicit source addr for strings)	DS	ES, SS, or CS
DI (implicit dest addr for strings)	ES	Never

8086/8088 Instructions

Notes for 8086/8088 Instructions

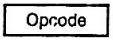
The individual instruction descriptions are shown by a format box such as the following:



These are byte-wise representations of the object code generated by the assembler and are interpreted as follows:

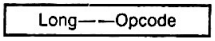
- Opcode is the 8-bit opcode for the instruction. The actual opcode generated is defined in the "Opcode" column of the instruction table that follows each format box.
- m/op/r/m is the byte that specifies the operands of the instruction. It contains a 2-bit mode field (m), a 3-bit register field (op), and a 3-bit register or memory (r/m) field.
- Dashed blank boxes following the m/op/r/m box are for any displacement required by the mode field.
- Data is for a byte of immediate data.
- A dashed blank box following a Data box is used whenever the immediate operand is a word quantity.

AAA = ASCII Adjust for Addition



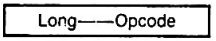
Opcode	Clocks	Operation
37	4	adjust AL, flags, AH

AAD = ASCII Adjust for Division



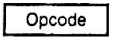
Opcode	Clocks	Operation
D5,0A	60	Adjust AL, AH prior to division

AAM = ASCII Adjust for Multiplication



Opcode	Clocks	Operation
D4,0A	83	Adjust AL, AH after multiplication

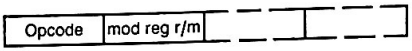
AAS = ASCII Adjust for Subtraction



Opcode	Clocks	Operation
3F	4	adjust AL, flags, AH

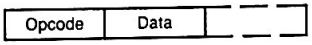
ADC = Integer Add with Carry

Memory/Reg + Reg



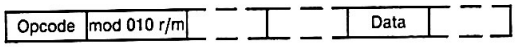
	Opcode	Clocks	Operation
Byte	12	3	Reg8 ← CF + Reg 8 + Reg8
	12	9+EA	Reg8 ← CF + Reg8 + Mem8
	10	16+EA	Mem8 ← CF + Mem8 + Reg8
Word	13	3	Reg16 ← CF + Reg16 + Reg16
	13	9+EA	Reg16 ← CF + Reg16 + Mem16
	11	16+EA	Mem16 ← CF + Mem16 + Reg16

Immed to AX/AL



	Opcode	Clocks	Operation
Byte	14	4	AL ← CF + AL + Immed8
Word	15	4	AX ← CF + AX + Immed16

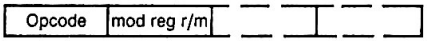
Immed to Memory/Reg



	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← CF + Reg8 + Immed8
	80	17+EA	Mem8 ← CF + Mem8 + Immed8
Word	81	4	Reg16 ← CF + Reg16 + Immed16
	81	17+EA	Mem16 ← CF + Mem16 + Immed16
	83	4	Reg16 ← CF + Reg16 + Immed8
	83	17+EA	Mem16 ← CF + Mem16 + Immed8

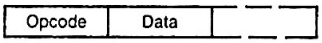
ADD = Integer Addition

Memory/Reg + Reg



	Opcode	Clocks	Operation
Byte	02	3	Reg8 ← Reg8 + Reg8
	02	9+EA	Reg8 ← Reg8 + Mem8
	00	16+EA	Mem8 ← Mem8 + Reg8
Word	03	3	Reg16 ← Reg16 + Reg16
	03	9+EA	Reg16 ← Reg16 + Mem16
	01	16+EA	Mem16 ← Mem16 + Reg16

Immed to AX/AL



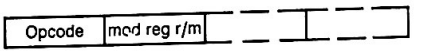
	Opcode	Clocks	Operation
	04	4	AL ← AL + Immed8
	05	4	AX ← AX + Immed16

Immed to Memory/Reg

	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 + Immed8
	80	17+EA	Mem8 ← Mem8 + Immed8
Word	81	4	Reg16 ← Reg16 + Immed16
	81	17+EA	Mem16 ← Mem16 + Immed16
	83	4	Reg16 ← Reg16 + Immed8
	83	17+EA	Mem16 ← Mem16 + Immed8

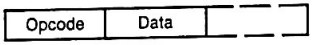
AND = Logical AND

Memory/Reg with Reg



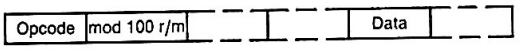
	Opcode	Clocks	Operation
Byte	22	3	Reg8 ← Reg8 AND Reg8
	22	9+EA	Reg8 ← Reg8 AND Mem8
	20	16+EA	Mem8 ← Mem8 AND Reg8
Word	23	3	Reg16 ← Reg16 AND Reg16
	23	9+EA	Reg16 ← Reg16 AND Mem16
	21	16+EA	Mem16 ← Mem16 AND Reg16

Immed to AX/AL



	Opcode	Clocks	Operation
Byte	24	4	AL ← AL AND Immed8
Word	25	4	AX ← AX AND Immed16

Immed to Memory/Reg



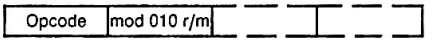
	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 AND Immed8
	80	17+EA	Mem8 ← Mem8 AND Immed8
Word	81	4	Reg16 ← Reg16 AND Immed16
	81	17+EA	Mem16 ← Mem16 AND Immed16

CALL = Call

Within segment or group, IP relative

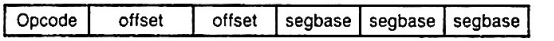
Opcode	Clocks	Operation
E8	19	IP ← IP + Disp16 — (SP) ← return link

Within segment or group, Indirect



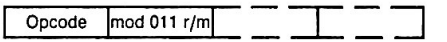
Opcode	Clocks	Operation
FF	16	IP ← Reg16 — (SP) ← return link
FF	21+EA	IP ← Mem16 — (SP) ← return link
FF	21+EA	IP ← Mem16 — (SP) ← return link

Inter-segment or group, Direct



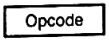
Opcode	Clocks	Operation
9A	28	CS ← segbase IP ← offset

Inter-segment or group, Indirect



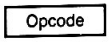
Opcode	Clocks	Operation
FF	37+EA	CS ← segbase IP ← offset

CBW = Convert Byte to Word



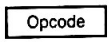
Opcode	Clocks	Operation
98	2	convert byte in AL to word in AX

CLC = Clear Carry Flag



Opcode	Clocks	Operation
F8	2	clear the carry flag

CLD = Clear Direction Flag

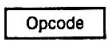


Opcode	Clocks	Operation
FC	2	clear direction flag

CLI = Clear Interrupt Enable Flag

Opcode	Clocks	Operation
FA	2	clear interrupt flag

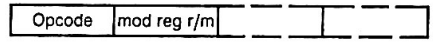
CMC = Complement Carry Flag



Opcode	Clocks	Operation
F5	2	complement carry flag

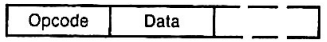
CMP = Compare Two Operands

Memory/Reg with Reg



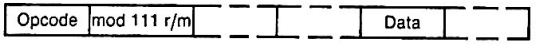
	Opcode	Clocks	Operation
Byte	38	3	flags ← Reg8 - Reg8
	38	9+EA	flags ← Reg8 - Mem8
	3A	9+EA	flags ← Mem8 - Reg8
Word	39	3	flags ← Reg16 - Reg16
	39	9+EA	flags ← Reg16 - Mem16
	3B	9+EA	flags ← Mem16 - Reg16

Immed to AX/AL



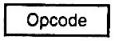
	Opcode	Clocks	Operation
Byte	3C	4	flags AL - Immed8
Word	3D	4	flags AX - Immed16

Immed to Memory/Reg



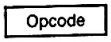
	Opcode	Clocks	Operation
Byte	80	4	flags ← Reg8 - Immed8
	80	10+EA	flags ← Mem8 - Immed8
Word	81	4	flags ← Reg16 - Immed16
	81	10+EA	flags ← Mem16 - Immed16
	83	4	flags ← Reg16 - Immed8
	83	10+EA	flags ← Mem16 - Immed8

CWD = Convert Word to Doubleword



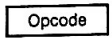
Opcode	Clocks	Operation
99	5	convert word in AX to doubleword in DX:AX

DAA = Decimal Adjust for Addition



Opcode	Clocks	Operation
27	4	adjust AL, flags, AH

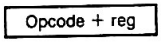
DAS = Decimal Adjust for Subtraction



Opcode	Clocks	Operation
2F	4	adjust AL, flags, AH

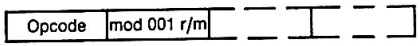
DEC = Decrement by 1

Word Register



Opcode	Clocks	Operation
48+reg	2	Reg16 ← Reg16 - 1

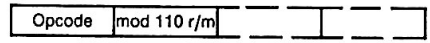
Memory/Byte Register



	Opcode	Clocks	Operation
Byte	FE	3	Reg8 ← Reg8 - 1
	FE	15+EA	Mem8 ← Mem8 - 1
Word	FF	15+EA	Mem16 ← Mem16 - 1

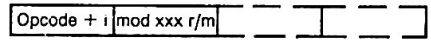
DIV = Unsigned Division

Memory/Reg with AX or DX:AX



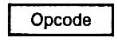
	Opcode	Clocks	Operation
Byte	F6	80-90	AH,AL ← AX / Reg8
	F6	(86-96)+EA	AH,AL ← AX / Mem8
Word	F7	144-162	DX,AX ← DX:AX / Reg16
	F7	(150-168)+EA	DX,AX ← DX:AX / Mem16

ESC = Escape



Opcode	Clocks	Operation
D8+i	8+EA	data bus ← (EA)
D8+i	2	data bus ← (EA)

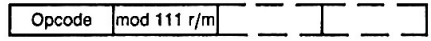
HLT = Halt



Opcode	Clocks	Operation
F4	2	halt operation

IDIV = Signed Division

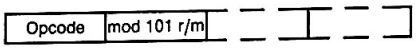
Memory/Reg with AX or DX:AX



	Opcode	Clocks	Operation
Byte	F6	101-112	AH,AL ← AX / Reg8
	F6	(107-118)+EA	AH,AL ← AX / Mem8
Word	F7	165-184	DX,AX ← DX:AX / Reg16
	F7	(171-190)+EA	DX,AX ← DX:AX / Mem16

IMUL = Signed Multiplication

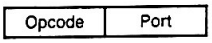
Memory/Reg with AL or AX



	Opcode	Clocks	Operation
Byte	F6	80-98	AX ← AL*Reg8
	F6	(86-104)+EA	AX ← AL*Mem8
Word	F7	128-154	DX:AX ← AX*Reg16
	F7	(134-160)+EA	DX:AX ← AX*Mem16

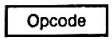
IN = Input Byte, Word

Fixed port



	Opcode	Clocks	Operation
Byte	E4	10	AL ← Port8
	E5	10	AX ← Port8

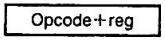
Variable port



	Opcode	Clocks	Operation
Word	EC	8	AL ← Port16(in DX)
	ED	8	AX ← Port16(in DX)

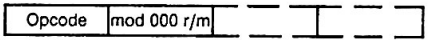
INC = Increment by 1

Word Register



Opcode	Clocks	Operation
40+reg	2	Reg16 ← Reg16 + 1

Memory/Byte Register

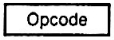


	Opcode	Clocks	Operation
Byte	FE	3	Reg8 ← Reg8 + 1
	FE	15+EA	Mem8 ← Mem8 + 1
Word	FF	15+EA	Mem16 ← Mem16 + 1

INT
INTO = Interrupt

Opcode	Clocks	Operation
CC	52	Interrupt 3
CD	51	Interrupt 'type'
CE	53 or 4	Interrupt4 if FLAGS.OF=1, else NOP

IRET = Return from Interrupt



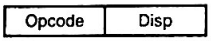
Opcode	Clocks	Operation
CF	24	Return from interrupt

Jcond = Jump on Condition

Operation

If condition is true then do;
  sign-extend displacement to 16 bits;
  IP ← IP + sign-extended displacement;
  end if;

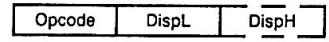
Format



Opcode	Clocks	Operation	*cond* =
77	16 or 4	jump if above	JA
73	16 or 4	jump if above or equal	JAE
72	16 or 4	jump if below	JB
76	16 or 4	jump if below or equal	JBE
72	16 or 4	jump if carry set	JC
74	16 or 4	jump if equal	JE
7F	16 or 4	jump if greater	JG
7D	16 or 4	jump if greater or equal	JGE
7C	16 or 4	jump if less	JL
7E	16 or 4	jump if less or equal	JLE
76	16 or 4	jump if not above	JNA
72	16 or 4	jump if neither above nor equal	JNAE
73	16 or 4	jump if not below	JNB
77	16 or 4	jump if neither below nor equal	JNBE
73	16 or 4	jump if no carry	JNC
75	16 or 4	jump if not equal	JNE
7E	16 or 4	jump if not greater	JNG
7C	16 or 4	jump if neither greater nor equal	JNGE
7D	16 or 4	jump if not less	JNL
7F	16 or 4	jump if neither less nor equal	JNLE
71	16 or 4	jump if no overflow	JNO
7B	16 or 4	jump if no parity	JNP
79	16 or 4	jump if positive	JNS
75	16 or 4	jump if not zero	JNZ
70	16 or 4	jump if overflow	JO
7A	16 or 4	jump if parity	JP
7A	16 or 4	jump if parity even	JPE
7B	16 or 4	jump if parity odd	JPO
78	16 or 4	jump if sign	JS
74	18 or 6	jump if zero	JZ
E3	18 or 6	jump if CX is zero (does not test flags)	JCXZ

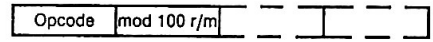
JMP = Jump

Within segment or group, IP relative



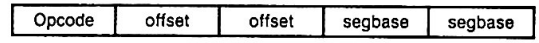
Opcode	Clocks	Operation
E9	15	IP ← IP + Disp16
EB	15	IP ← IP + Disp8 (Disp8 sign-extended)

Within segment or group, Indirect



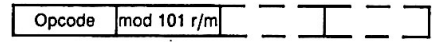
Opcode	Clocks	Operation
FF	11	IP ← Reg16
FF	18+EA	IP ← Mem16
FF	18+EA	IP ← Mem16

Inter-segment or group, Direct



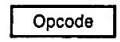
Opcode	Clocks	Operation
EA	15	CS ← segbase IP ← offset

Inter-segment or group, Indirect



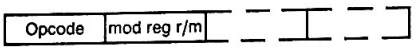
Opcode	Clocks	Operation
FF	24+EA	CS ← segbase IP ← offset

LAHF = Load AH from Flags



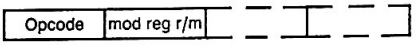
Opcode	Clocks	Operation
9F	4	copy low byte of flags word to AH

LDS/LES = Load Pointer to DS/ES and Register



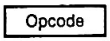
Opcode	Clocks	Operation
C4	16+EA	dword pointer at EA goes to reg16 (1st word) and ES (2nd word)
C5	16+EA	dword pointer at EA goes to reg16 (1st word) and DS (2nd word)

LEA = Load Effective Address



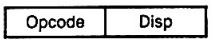
Opcode	Clocks	Operation
8D	2+EA	Reg16←EA

LOCK = Assert Bus Lock



Opcode	Clocks	Operation
F0	2	assert the bus lock next instruction

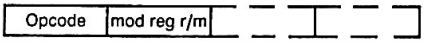
LOOP*xx* = Loop Control



Opcode	Clocks	Operation	*xx* =
E1	18 or 6	dec CX; loop if equal and CX not 0	LOOPE
E0	19 or 5	dec CX; loop if not equal and CX not 0	LOOPNE
E1	18 or 6	dec CX; loop if zero and CX not 0	LOOPZ
E0	19 or 5	dec CX; loop if not zero and CX not 0	LOOPNZ
E2	17 or 5	dec CX; loop if CX not 0	LOOP

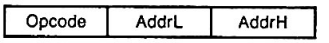
MOV = Move Data

Memory/Reg to or from Reg



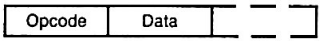
	Opcode	Clocks	Operation
Byte	88	9+EA	Mem8←Reg8
	88	2	Reg8←Reg8
	8A	8+EA	Reg8←Mem8
Word	89	9+EA	Mem16←Reg16
	89	2	Reg16←Reg16
	8B	8+EA	Reg16←Mem16

Direct-Addressed Memory to or from AX/AL



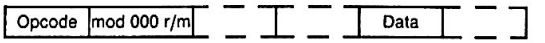
	Opcode	Clocks	Operation
Byte	A0	10	AL←Mem8
	A2	10	Mem8←AL
Word	A1	10	AX←Mem16
	A3	10	Mem16←AX

Immed to Reg



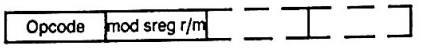
	Opcode	Clocks	Operation
Byte	B0+reg	4	Reg 8←Immed8
Word	B8+reg	4	Reg16←Immed16

Immed to Memory/Reg



Opcode	Clocks	Operation
C6	4	Reg8←Immed8
C6	10+EA	Mem8←Immed8
C7	4	Reg16←Immed16
C7	10+EA	Mem16←Immed16

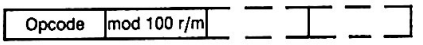
Memory/Reg to or from SReg



	Opcode	Clocks	Operation
Word	8C	9+EA	Mem16←SReg
	8C	2	Reg16←SReg
	8E	8+EA	SReg←Mem16
	8E	2	SReg←Reg16

MUL = Unsigned Multiplication

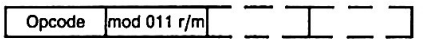
Memory/Reg with AL or AX



	Opcode	Clocks	Operation
Byte	F6	70-77	AX←AL*Reg8
	F6	(76-83)+EA	AX←AL*Mem8
Word	F7	118-133	DX:AX←AX*Reg16
	F7	(124-139)+EA	DX:AX←AX*Mem16

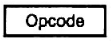
NEG = Negate an Integer

Memory/Reg



Opcode	Clocks	Operation
F6	3	Reg8←00H - Reg 8
F7	3	Reg16←0000H - Reg16
F6	16+EA	Mem8←00H - Mem8
F7	16+EA	Mem16←0000H - Mem16

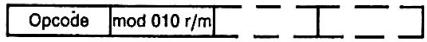
NOP = No Operation



Opcode	Clocks	Operation
90	3	no operation

NOT = Form One's Complement

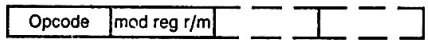
Memory/Reg



	Opcode	Clocks	Operation
Byte	F6	3	Reg8←0FFH - Reg8
	F6	16+EA	Mem8←0FFH - Mem8
Word	F7	3	Reg16←0FFFFH - Reg16
	F7	16+EA	Mem16←0FFFFH - Mem16

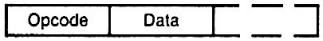
OR = Logical Inclusive OR

Memory/Reg with Reg



	Opcode	Clocks	Operation
Byte	0A	3	Reg8←Reg8 OR Reg8
	0A	9+EA	Reg8←Reg8 OR Mem8
	08	16+EA	Mem8←Mem8 OR Reg8
Word	0B	3	Reg16←Reg16 OR Reg 16
	0B	9+EA	Reg16←Reg16 OR Mem16
	09	16+EA	Mem16←Mem16 OR Reg16

Immed to AX/AL



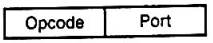
Opcode	Clocks	Operation
0C	4	AL←AL OR Immed8
0D	4	AX←AX OR Immed16

Immed to Memory/Reg

	Opcode	Clocks	Operation
Byte	80	4	Reg8←Reg8 OR Immed8
	80	17+EA	Mem8←Mem8 OR Immed8
Word	81	4	Reg16←Reg16 OR Immed16
	81	17+EA	Mem16←Mem16 OR Immed16

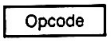
OUT = Output Byte, Word

Fixed port



	Opcode	Clocks	Operation
Byte	E6	10	Port8 ← AL
	E7	10	Port8 ← AX

Variable port



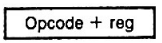
	Opcode	Clocks	Operation
Word	EE	8	Port16 (in DX) ← AL
	EF	8	Port16 (in DX) ← AX

POP = Pop a Word from the Stack

Word Memory

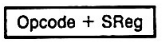
Opcode	Clocks	Operation
8F	17+EA	Mem16 ← (SP)++

Word Register



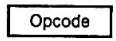
Opcode	Clocks	Operation
58+reg	8	Reg16 ← (SP)++

Segment Register



Opcode	Clocks	Operation
07+SReg	8	SReg ← (SP)++

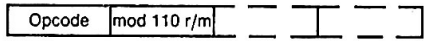
POPF = Pop the TOS into the Flags



Opcode	Clocks	Operation
9D	8	FLAGS ← (SP)++

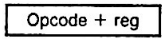
PUSH = Push a Word onto the Stack

Memory/Reg



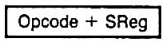
Opcode	Clocks	Operation
FF	16+EA	—(SP) ← Mem16

Word Register



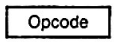
Opcode	Clocks	Operation
50+reg	11	—(SP) ← Reg16

Segment Register



Opcode	Clocks	Operation
06+SReg	10	—(SP) ← SReg

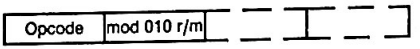
PUSHF = Push the Flags to the Stack



Opcode	Clocks	Operation
9C	10	—(SP) ← FLAGS

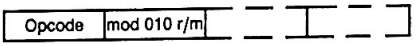
RCL = Rotate Left Through Carry

Memory or Reg by 1



	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg 8 by 1
	D0	15+EA	rotate Mem8 by 1
Word	D1	2	rotate Reg 16 by 1
	D1	15+EA	rotate Mem16 by 1

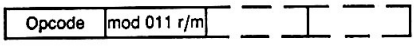
Memory or Reg by count in CL



	Opcode	Clocks	Operation
Byte	D2	8+4/bit	rotate Reg8 by CL
	D2	20+EA+4/bit	rotate Mem8 by CL
Word	D3	8+4/bit	rotate Reg16 by CL
	D3	20+EA+4/bit	rotate Mem16 by CL

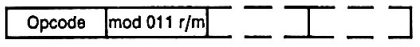
RCR = Rotate Right Through Carry

Memory or Reg by 1



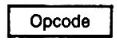
	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg8 by 1
	D0	15+EA	rotate Mem8 by 1
Word	D1	2	rotate Reg16 by 1
	D1	15+EA	rotate Mem16 by 1

Memory or Reg by count in CL



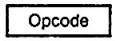
	Opcode	Clocks	Operation
Byte	D2	8+4/bit	rotate Reg8 by CL
	D2	20+EA+4/bit	rotate Mem8 by CL
Word	D3	8+4/bit	rotate Reg16 by CL
	D3	20+EA+4/bit	rotate Mem16 by CL

REP*x* = Repeat Prefix



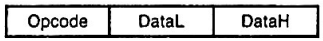
Opcode	Clocks	Operation	REP*x* =
F3	2	repeat next instruction until CX=0	REP
F3	2	repeat next instruction until CX=0 or ZF=1	REPE REPZ
F2	2	repeat next instruction until CX=0 or ZF=0	REPNE REPNZ

RET = Return from Subroutine



Opcode	Clocks	Operation
C3	8	intra-segment return
CB	18	inter-segment return

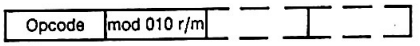
Return and add constant to SP



Opcode	Clocks	Operation
C2	12	intra-segment ret and add
CA	17	inter-segment ret and add

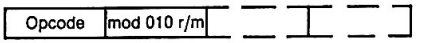
ROL = Rotate Left

Memory or Reg by 1



	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg8 by 1
	D0	15+EA	rotate Mem8 by 1
Word	D1	2	rotate Reg16 by 1
	D1	15+EA	rotate Mem16 by 1

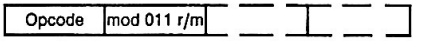
Memory or Reg by count in CL



	Opcode	Clocks	Operation
Byte	D2	8+4/bit	rotate Reg8 by CL
	D2	20+Ea+4/bit	rotate Mem8 by CL
Word	D3	8+4/bit	rotate Reg16 by CL
	D3	20+EA+4/bit	rotate Mem16 by CL

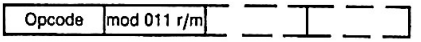
ROR = Rotate Right

Memory or Reg by 1



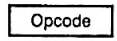
	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg8 by 1
	D0	15+EA	rotate Mem8 by 1
Word	D1	2	rotate Reg16 by 1
	D1	15+EA	rotate Mem16 by 1

Memory or Reg by count in CL



	Opcode	Clocks	Operation
Byte	D2	8+4/bit	rotate Reg8 by CL
	D2	20+EA+4/bit	rotate Mem8 by CL
	D3	8+4/bit	rotate Reg16 by CL
	D3	20+EA+4/bit	rotate Mem16 by CL

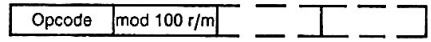
SAHF = Store AH in Flags



Opcode	Clocks	Operation
9E	4	copy AH to low byte of flags word

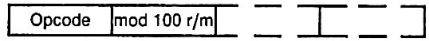
SAL/SHL = Arithmetic/Logical Left Shift

Memory or Reg by 1



	Opcode	Clocks	Operation
Byte	D0	2	shift Reg8 by 1
	D0	15+EA	shift Mem8 by 1
Word	D1	2	shift Reg16 by 1
	D1	15+EA	shift Mem16 by 1

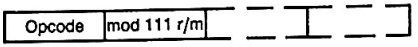
Memory or Reg by count in CL



	Opcode	Clocks	Operation
Byte	D2	8+4/bit	shift Reg8 by CL
	D2	20+EA+4/bit	shift Mem8 by CL
Word	D3	8+4/bit	shift Reg16 by CL
	D3	20+EA+4/bit	shift Mem16 by CL

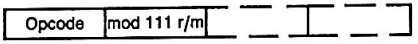
SAR = Arithmetic Right Shift

Memory or Reg by 1



	Opcode	Clocks	Operation
Byte	D0	2	shift Reg8 by 1
	D0	15+EA	shift Mem8 by 1
Word	D1	2	shift Reg16 by 1
	D1	15+EA	shift Mem16 by 1

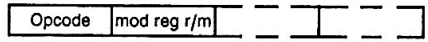
Memory or Reg by count in CL



	Opcode	Clocks	Operation
Byte	D2	8+4/bit	shift Reg8 by CL
	D2	20+EA+4/bit	shift Mem8 by CL
Word	D3	8+4/bit	shift Reg16 by CL
	D3	20+EA+4/bit	shift Mem16 by CL

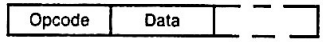
SBB = Integer Subtraction with Borrow

Memory/Reg with Reg



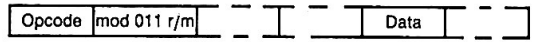
	Opcode	Clocks	Operation
Byte	1A	3	Reg8 ← Reg8 - Reg8 - CF
	1A	9+EA	Reg8 ← Reg8 - Mem8 - CF
	18	16+EA	Mem8 ← Mem8 - Reg8 - CF
Word	1B	3	Reg16 ← Reg16 - Reg16 - CF
	1B	9+EA	Reg16 ← Reg16 - Mem16 - CF
	19	16+EA	Mem16 ← Mem16 - Reg16 - CF

Immed from AX/AL



Opcode	Clocks	Operation
1C	4	AL ← AL - Immed8 - CF
1D	4	AX ← AX - Immed16 - CF

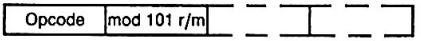
Immed from Memory/Reg



Opcode	Clocks	Operation
80	4	Reg8 ← Reg8 - Immed8 - CF
80	17+EA	Mem8 ← Mem8 - Immed8 - CF
81	4	Reg16 ← Reg16 - Immed16 - CF
81	17+EA	Mem16 ← Mem16 - Immed16 - CF
83	4	Reg16 ← Reg16 - Immed8 - CF
83	17+EA	Mem16 ← Mem16 - Immed8 - CF (Immed8 is sign-extended before subtract)

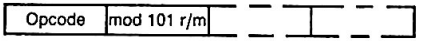
SHR = Logical Right Shift

Memory or Reg by 1



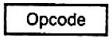
	Opcode	Clocks	Operation
Byte	D0	2	shift Reg8 by 1
	D0	15+EA	shift Mem8 by 1
Word	D1	2	shift Reg16 by 1
	D1	15+EA	shift Mem16 by 1

Memory or Reg by count in CL



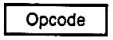
	Opcode	Clocks	Operation
Byte	D2	8+4/bit	shift Reg8 by CL
	D2	20+Ea+4/bit	shift Mem8 by CL
Word	D3	8+4/bit	shift Reg16 by CL
	D3	20+EA+4/bit	shift Mem16 by CL

STC = Set Carry Flag



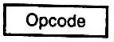
Opcode	Clocks	Operation
F9	2	set the carry flag

STD = Set Direction Flags



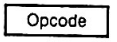
Opcode	Clocks	Operation
FD	2	set direction flag

STI = Set Interrupt Enable Flag



Opcode	Clocks	Operation
FB	2	set interrupt flag

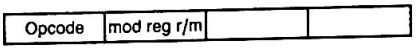
String = String Operations



Opcode	Clocks	Operation	*String* =
A6	22	flags ← (SI) - (DI)	CMPS
A7	22	flags ← (SI) - (DI)	CMPS
A4	18	(DI) ← (SI)	MOVS
A5	18	(DI) ← (SI)	MOVS
AE	15	flags ← (DI) - AL	SCAS
AF	15	flags ← (DI) - AX	SCAS
AC	12	AL ← (SI)	LODS
AD	12	AX ← (SI)	LODS
AA	11	(DI) ← AL	STOS
AB	11	(DI) ← AX	STOS

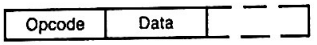
SUB = Integer Subtraction

Memory/Reg with Reg



	Opcode	Clocks	Operation
Byte	2A	3	Reg8 Reg8 - Reg8
	2A	9+EA	Reg8 Reg8 - Mem8
	28	16+EA	Mem8 Mem8 - Reg8
Word	2B	3	Reg16 Reg16 - Reg16
	2B	9+EA	Reg16 Reg16 - Mem16
	29	16+EA	Mem16 Mem16 - Reg16

Immed to AX/AL



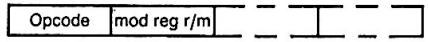
	Opcode	Clocks	Operation
Byte	2C	4	AL ← AL - Immed8
Word	2D	4	AX ← AX - Immed16

Immed to Memory/Reg

	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 - Immed8
	80	17+EA	Mem8 ← Mem8 - Immed8
Word	81	4	Reg16 ← Reg16 - Immed16
	81	17+EA	Mem16 ← Mem16 - Immed16
	83	4	Reg16 ← Reg16 - Immed8
	83	17+EA	Mem16 ← Mem16 - Immed8

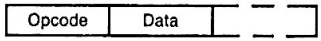
TEST = Logical Compare

Memory/Reg with Reg



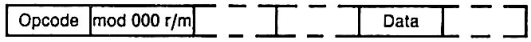
	Opcode	Clocks	Operation
Byte	84	3	flags ← Reg8 AND Reg8
	84	9+EA	flags ← Reg8 AND Mem8
Word	85	3	flags ← Reg16 AND Reg16
	85	9+EA	flags ← Reg16 AND Mem16

Immed to AX/AL



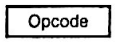
	Opcode	Clocks	Operation
Byte	A8	4	flags ← AL AND Immed8
Word	A9	4	flags ← AX AND Immed16

Immed to Memory/Reg



	Opcode	Clocks	Operation
Byte	F6	5	flags ← Reg8 AND Immed8
	F6	11+EA	flags ← Mem8 AND Immed8
Word	F7	5	flags ← Reg16 AND Immed16
	F7	11+EA	flags ← Mem16 AND Immed16

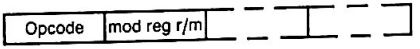
WAIT = Wait While TEST Pin Not Asserted



Opcode	Clocks	Operation
9B	3+5n	none

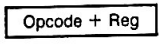
XCHG = Exchange Memory/Register with Register

Memory/Reg with Reg



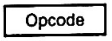
	Opcode	Clocks	Operation
Byte	86	4	Reg8 ↔ Reg8
	86	17+EA	Mem8 ↔ Mem8
Word	87	4	Reg16 ↔ Reg16
	87	17+EA	Mem16 ↔ Mem16

Word Register with AX



Opcode	Clocks	Operation
90+Reg	3	AX ↔ Reg16

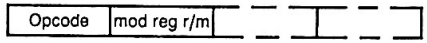
XLAT XLATB = Table Look-up Translation



Opcode	Clocks	Operation
D7	11	replace AL with table entry

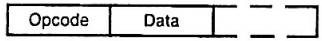
XOR = Logical Exclusive OR

Memory/Reg with Reg



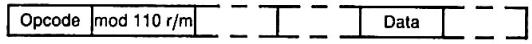
	Opcode	Clocks	Operation
Byte	32	3	Reg8 ← Reg8 XOR Reg8
	32	9+EA	Reg8 ← Reg8 XOR Mem8
	30	16+EA	Mem8 ← Mem8 XOR Reg8
Word	33	3	Reg16 ← Reg16 XOR Reg16
	33	9+EA	Reg16 ← Reg16 XOR Mem16
	31	16+EA	Mem16 ← Mem16 XOR Reg16

Immed to AX/AL



Opcode	Clocks	Operation
34	4	AL ← AL XOR Immed8
35	4	AX ← AX XOR Immed16

Immed to Memory/Reg



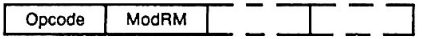
	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 XOR Immed8
	80	17+EA	Mem8 ← Mem8 XOR Immed8
Word	81	4	Reg16 ← Reg16 XOR Immed16
	81	17+EA	Mem16 ← Mem16 XOR Immed16

186 INSTRUCTIONS

Notes for iAPX 186 Instructions

These instructions can be used only if the MOD186 control is specified. When MOD186 is specified, clocks for all instructions are as stated under "Clocks for MOD186 Operation."

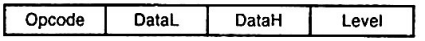
BOUND = Check Array Against Bounds



Opcode Operation

62 if Reg16<Mem16 at EA, or Reg16>Mem16 at EA+2 then INTERRUPT 5

ENTER = High Level Procedure Entry



Opcode Operation

C8 build new stack frame

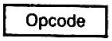
IMUL = Signed Multiplication

Mem/Reg* Immediate to Reg

Opcode Operation

6B Reg 16 ← Reg 16 * Immed 8
6B Reg 16 ← Reg 16 * Immed 8
6B Reg 16 ← Mem 16 * Immed 8
69 Reg 16 ← Reg 16 * Immed 16
69 Reg 16 ← Reg 16 * Immed 16
69 Reg 16 ← Mem 16 * Immed 16

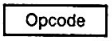
LEAVE = High Level Procedure Exit



Opcode Operation

C9 release current stack frame and return to prior frame.

POPA = Pop All Registers

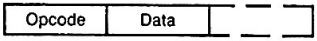


Opcode Operation

61 restore registers from stack

PUSH = Push a Word onto the Stack

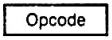
Word Immediate



Opcode Operation

6A —(SP) ← Immed8 (sign extended)
68 —(SP) ← Immed16

PUSHA = Push All Registers

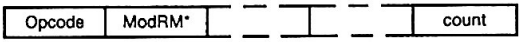


Opcode Operation

60 save registers on the stack

RCL = Rotate Left Through Carry

Mem or Reg by Immed8

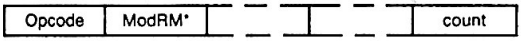


*—(Reg field = 011)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

RCR = Rotate Right Through Carry

Mem or Reg by Immed8

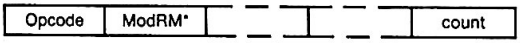


*—(Reg field = 011)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

ROL = Rotate Left

Mem or Reg by Immed8

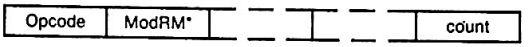


*—(Reg field = 000)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

ROR = Rotate Right

Mem or Reg by Immed8

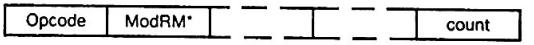


*—(Reg field = 001)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

SAL/SHL = Arithmetic/Logical Left Shift

Mem or Reg by immediate count

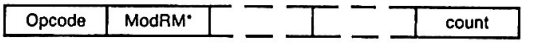


*—(Reg field = 100)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

SAR = Arithmetic Right Shift

Mem or Reg by Immed8

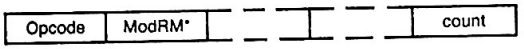


*—(Reg field = 111)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

SHR = Logical Right Shift

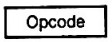
Mem or Reg by Immed8



*—(Reg field = 101)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

String = String Operations (INS/OUTS)

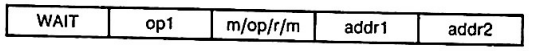


Opcode	Clocks	Operation
6E	INS	(DI) → port(DX)
6F	INS	(DI) → port(DX:DX+1)
6C	OUTS	port(DX) → (SI)
6D	OUTS	port(DX:DX+1) → (SI)

8087 INSTRUCTIONS

Notes for 8087 Instructions

The individual instruction descriptions are shown by a format box such as the following:



These are the byte-wise representations of the object code generated by the assembler and are interpreted as follows:

- WAIT is an 8086 wait instruction, NOP or emulator instruction.
- op1 is the opcode, possibly taking two bytes.
- m/op/r/m byte (middle 3-bits is part of the opcode).
- addr1 and addr2 are offsets of either 8 or 16 bits.

For integer functions, m = 0 for short-integer memory operand; 1 for word-integer memory operand.
For real functions, m = 0 for short-real memory operand; 1 for longreal memory operand.
i = stack element index.

If mod = 00 then DISP = 0 , disp-lo and disp-hi are absent.
If mod = 01 then DISP = disp-lo sign-extended to 16 bits, disp-hi is absent.
If mod = 10 then DISP = disp-hi; disp-lo.
If mod = 11 then r/m is treated as an ST(i) field.

If r/m = 000 then EA = (BX)+(SI)+DISP
If r/m = 001 then EA = (BX)+(DI)+DISP
If r/m = 010 then EA = (BP)+(SI)+DISP
If r/m = 011 then EA = (BP)+(DI)+DISP
If r/m = 100 then EA = (SI)+DISP
If r/m = 101 then EA = (DI)+DISP
If r/m = 110 then EA (BP)+DISP*
If r/m = 111 then EA = (BX)+DISP
*Except if mod = 000 and r/m = 110 then EA = disp-hi; disp-lo.

ST(0) = Current stack top
ST(i) = ith register below stack top
d = Destination
0 — Destination is ST(0)
1 — Destination is ST(i)
P = Pop
0 — No pop
1 — Pop ST(0)
R = Reverse
0 — Destination (op) source
1 — Source (op) destination

For FSQRT: $-0 \leq ST(0) \leq +\infty$
For FSCALE: $-2^{15} \leq ST(1) < +2^{15}$ and ST(1) integer
For F2XM1: $0 \leq ST(0) \leq 2^{-1}$
For FYL2X: $0 \leq ST(0) < \infty$
$-\infty < ST(1) < +\infty$
For FYL2XP1: $0 < |ST(0)| < (2-\sqrt{2})/2$
$-\infty \leq ST(1) < \infty$
For FPTAN: $0 \leq ST(0) < \pi/4$
For FPATAN: $0 \leq ST(0) < ST(1) < +\infty$

F2XMI = Compute $2^x - 1$

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F0	CD 19 F0	500 310-630	ST ← $2^{ST}-1$

FABS = Absolute Value

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 E1	CD 19 E1	14 10-17	ST ← |ST|

FADD = Add Real

Stack top + Stack element

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 C0+i	CD 18 C0+i	85 70-100	ST ← ST + ST(i)
9B DC C0+i	CD 1C C0+i	85 70-100	ST(i) ← ST + ST(i)

Stack top + memory operand

WAIT	op1	mod 000 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 m0rm	CD 18 m0rm	105+EA (90-120)+EA	ST ← ST + mem-op (short-real)
9B DC m0rm	CD 1C m0rm	110+EA (95-125)+EA	ST ← ST + mem-op (long-real)

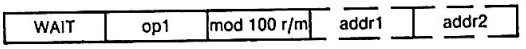
FADDP = Add Real and Pop

Stack top + Stack Element

WAIT	op1	op2 + i

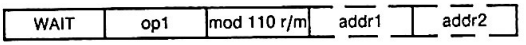
8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DE C1	CD 1E C1	90 75-105	ST(1) ← ST + ST(1) pop stack
9B DE C0+i	CD 1E C0+i	90 75-105	ST(i) ← ST + ST(i) pop stack

FBLD = Packed Decimal (BCD) Load



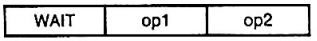
8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DF m4rm	CD 1F m4rm	300+EA (290-310)+EA	push stack ST ← mem-op

FBSTP = Packed Decimal (BCD) Store and Pop



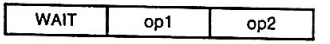
8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DF m6rm	CD 1F m6rm	530+EA (520-540)+EA	mem-op ← ST pop stack

FCHS = Change Sign



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 E0	CD 19 E0	15 10-17	ST ← —ST

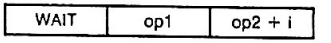
FCLEX
FNCLEX = Clear Exceptions



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DB E2	CD 1B E2	5 2-8	clear 8087 exceptions
90 DB E2	CD 1B E2	5 2-8	clear 8087 exceptions (no wait)

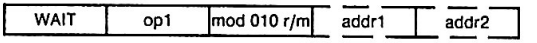
FCOM = Compare Real

Compare Stack top and Stack element



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 D1	CD 18 D1	45 40-50	ST — ST(1)
9B D8 D0+i	CD 18 D0+i	45 40-50	ST — ST(i)

Compare Stack top and memory operands



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 m2rm	CD 18 m2rm	65+EA (60-70)+EA	ST — memop (short-real)
9B DC m2rm	CD 1C m2rm	70+EA (65-75)+EA	ST — memop (long-real)

FCOMP = Compare Real and Pop

Compare Stack top and Stack element and pop

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 D9	CD 18 D9	47 42-52	ST — ST(1) pop stack
9B D8 D8+i	CD 18 D8+i	47 42-52	ST — ST(i) pop stack

Compare Stack top and memory operand and pop

WAIT	op1	mod 011 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 m3rm	CD 18 m3rm	68+EA (63-73)+EA	ST — mem-op pop stack (short-real)
9B DC m3rm	CD 1C m3rm	72+EA (67-77)+EA	ST — mem-op pop stack (long-real)

FCOMPP = Compare Real and Pop Twice

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DE D9	CD 1E D9	50 45-55	ST — ST(1) pop stack pop stack

FDECSTP = Decrement Stack Pointer

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F6	CD 19 F6	9 6-12	stack pointer ← stack pointer 1

FDISI
FNDISI = Disable Interrupts

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DB E1	CD 1B E1	5 2-8	Set 8087 interrupt mask
90 DB E1	CD 1B E1	5 2-8	Set 8087 interrupt mask (no wait)

FDIV = Divide Real

Stack top and Stack element

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 F0+i	CD 18 F0+i	198 193-203	ST ← ST/ST(i)
9B DC F8+i	CD 1C F8+i	198 193-203	ST(i) ← ST(i)/ST

Stack top and memory operand

WAIT	op1	mod 110 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 m6rm	CD 18 m6rm	220+EA (215-225)+EA	ST ← ST/mem-op (short-real)
9B DC m6rm	CD 1C m6rm	225+EA (220-230)+EA	ST ← ST/mem-op (long-real)

FDIVP = Divide Real and Pop

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DE F9	CD 1E F9	202 197-207	ST(1) ← ST(1)/ST pop stack
9B DE F8+i	CD 1E F8+i	202 197-207	ST(i) ← ST(i)/ST pop stack

FDIVR = Divide Real Reversed

Stack top and Stack element

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 F8+i	CD 18 F8+i	199 194-204	ST ← ST(i)/ST
9B DC F0+i	CD 1C F0+i	199 194-204	ST(i) ← ST/ST(i)

Stack top and memory operand

WAIT	op1	mod 111 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 m7rm	CD 18 m7rm	221+EA (216-226)+EA	ST ← mem-op/ST (short-real)
9B DC m7rm	CD 1C m7rm	226+EA (221-231)+EA	ST ← mem-op/ST (long-real)

FDIVRP = Divide Real Reversed and Pop

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DE F1	CD 1E F1	203 198-208	ST(1) ← ST/ST(1) pop stack
9B DE F0+i	CD 1E F0+i	203 198-208	ST(i) ← ST/ST(i)

FENI
FNENI = Enable Interrupts

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DB E0	CD 1B E0	5 2-8	clear 8087 interrupt mask
90 DB E0	CD 1B E0	5 2-8	clear 8087 interrupt mask (no wait)

FFREE = Free Register

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DD C0+i	CD 1D C0+i	11 9-16	TAG(i) masked empty

FIADD = Integer Add

WAIT	op1	mod 000 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DA m0rm	CD 1A m0rm	125+EA (108-143)+EA	ST ← ST + mem-op (short integer)
9B DE m0rm	CD 1E m0rm	120+EA (102-137)+EA	ST ← ST + mem-op (word integer)

FICOM = Integer Compare

WAIT	op1	mod 010 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DA m2rm	CD 1A m2rm	85+EA (78-91)+EA	ST — mem-op (short integer)
t9B DE m2rm	CD 1E m2rm	80+EA (72-86)+EA	ST — mem-op (word integer)

FICOMP = Integer Compare and Pop

WAIT	op1	mod 011 r/m	addr1	addr2

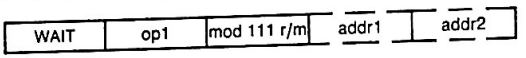
8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DA m3rm	CD 1A m3rm	87+EA (80-93)+EA	ST — mem-op pop stack (short integer)
9B DE m3rm	CD 1E m3rm	82+EA (74-88)+EA	ST — mem-op pop stack (word integer)

FIDIV = Integer Divide

WAIT	op1	mod 110 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DA m6rm	CD 1A m6rm	236+EA (230-243)+EA	ST ← ST/mem-op (short integer)
9B DE m6rm	CD 1E m6rm	230+EA (224-238)+EA	ST ← ST/mem-op (word integer)

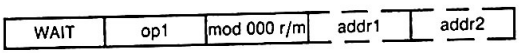
FIDIVR = Integer Divide Reversed



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DA m7rm	CD 1A m7rm	237+EA (231-245)+EA	ST←mem-op/ST (short integer)
9B DE m7rm	CD 1E m7rm	230+EA (225-239)+EA	ST←mem-op/ST (word integer)

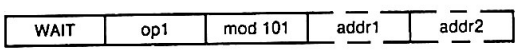
FILD = Integer Load

Word Integer or Short Integer



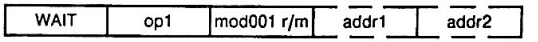
8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DB m0rm	CD 1B m0rm	56+EA (52-60)+EA	push stack ST←mem-op (short integer)
9B DF m0rm	CD 1F m0rm	50+EA (46-54)+EA	push stack ST←mem-op (word integer)

Long Integer



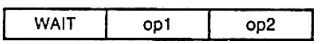
8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DF m5rm	CD 1F m5rm	64+EA (60-68)+EA	push stack ST←mem-op (long integer)

FIMUL = Integer Multiply



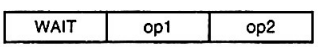
8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DA m1rm	CD 1A m1rm	136 + EA (130-144)+EA	ST←ST * mem-op (short integer)
9B DE m1rm	CD 1E m1m	130 + EA (124-138)+EA	ST←ST * mem-op (word integer)

FINCSTP = Increment Stack Pointer



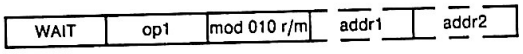
8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F7	CD 19 F7	9 6-12	stack pointer ← stack pointer + 1

FINIT FNINIT = Initialize Processor



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DB E3	CD 1B E3	5 2-8	initialize 8087
90 DB E3	CD 1B E3	5 2-8	initialize 8087 (no wait)

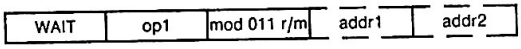
FIST = Integer Store



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DB m2rm	CD 1B m2rm	88+EA (82-92)+EA	mem-op → ST (short integer)
9B DF m2rm	CD 1F m2rm	86+EA (80-90)+EA	mem-op → ST (word integer)

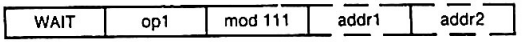
FISTP = Integer Store and Pop

Short Integer or Word Integer



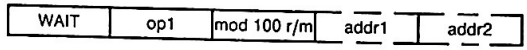
8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DB m3rm	CD 1B m3rm	90+EA (84-94)+EA	mem-op → ST pop stack (short integer)
9B DF m3rm	CD 1F m3rm	88+EA (82-92)+EA	mem-op → ST pop stack (word integer)

Long Integer



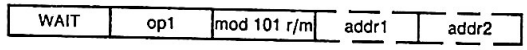
8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DF m7rm	CD 1F m7rm	100+EA (94-105)+EA	mem-op → ST pop stack (long integer)

FISUB = Integer Subtract



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DA m4rm	CD 1A m4rm	125+EA (108-143)+EA	ST → ST — mem-op (short integer)
9B DE m4rm	CD 1E m4rm	120+EA (102-137)+EA	ST → ST — mem-op (word integer)

FISUBR = Integer Subtract Reversed



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DA m5rm	CD 1A m5rm	125+EA (109-144)+EA	ST → mem-op — ST (short integer)
9B DE m5rm	CD 1E m5rm	120+EA (103-139)+EA	ST → mem-op — ST (word integer)

FLD = Load Real

Stack element to Stack top

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 C0+i	CD 19 C0+i	20 17-22	T, ← ST(i) push stack ST ← T,

Memory operand to Stack top
Short Integer or Long Integer

WAIT	op1	mod 000 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 m0rm	CD 19 m0rm	43+EA (38-56)+EA	push stack ST ← mem-op (short integer)
9B DD m0rm	CD 1D m0rm	46+EA (40-60)+EA	push stack ST ← mem-op (long integer)

Temp Real

WAIT	op1	mod 101	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DB m5rm	CD 1B m5rm	57+EA (53-65)+EA	push stack ST ← mem-op (temp real)

FLD1 = Load + 1.0

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 E8	CD 19 E8	18 15-21	push stack ST ← 1.0

FLDCW = Load Control Word

WAIT	op1	mod 101 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 m5rm	CD 19 m5rm	10+EA (7-14)+EA	processor control word ← mem-op

FLDENV = Load Environment

WAIT	op1	mod 100 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 m4rm	CD 19 m4rm	40+EA (35-45)+EA	8087 environment ← mem-op

FLDL2E = Load Log_2e

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 EA	CD 19 EA	18 15-21	push stack ST ← log_2e

FLDL2T = Load $Log_2 10$

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 E9	CD 19 E9	19 16-22	push stack ST ← $log_2 10$

FLDLG2 = Load $Log_{10} 2$

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 EC	CD 19 EC	21 18-24	push stack ST ← $log_{10} 2$

FLDPI = Load π

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 EB	CD 19 EB	19 16-22	push stack ST ← π

FLDZ = Load + 0.0

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 EE	CD 19 EE	14 11-17	push stack ST ← 0.0

FMUL = Multiply Real

Stack top and Stack element

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 C8+i	CD 18 C8+i	138 130-145	ST ← ST * ST(i)
9B DC C8+i	CD 1C C8+i	138 130-145	ST(i) ← ST(i) — ST

Stack top and memory operand

WAIT	op1	mod 001 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 m1rm	CD 18 m1rm	118+EA (110-125)+EA	ST ← ST * mem-op (short real)
9B DC m1rm	CD 1C m1rm	161+EA (154-168)+EA	ST ← ST * mem-op (long real)

FMULP = Multiply Real and Pop

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DE C9 +i	CD 1E C9+i	142 134-148	ST(i) ← ST(i) * ST pop stack

FNOP = No Operation

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 D0	CD 19 D0	13 10-16	ST ← ST

FPATAN = Partial Arctangent

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F3	CD 19 F3	650 250-800	T_1 ← arctan (ST(1)/ST) pop stack ST ← T_1

FPREM = Partial Remainder

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F8	CD 19 F8	125 15-190	ST ← REPEAT (ST — ST(1))

FPTAN = Partial Tangent

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F2	CD 19 F2	450 30-540	Y/X ← TAN (ST) ST ← Y push stack ST ← X

FRNDINT = Round to Integer

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 FC	CD 19 FC	45 16-50	ST ← nearest integer (ST)

FRSTOR = Restore Saved State

WAIT	op1	mod 100 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DD m4rm	CD 1D m4rm	202+EA (197-207)+EA	8087 state ← mem-op

FSAVE
FNSAVE = Save State

WAIT	op1	mod 110 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DD m6rm	CD 1D m6rm	202+EA (197-207)+EA	mem-op ← 8087 state
90 DD m6rm	CD 1D m6rm	202+EA (197-207)+EA	mem-op ← 8087 state (no wait)

FSCALE = Scale

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 FD	CD 19 FD	35 32-38	$ST \leftarrow ST * 2^{ST(0)}$

FSQRT = Square Root

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 FA	CD 19 FA	183 180-186	$ST \leftarrow \sqrt{ST}$

FST = Store Real

Stack top to Stack element

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DD D0+i	CD 1D D0+i	18 15-22	ST(i) ← ST

Stack top to memory operand

WAIT	op1	mod 010 r/m	addr1	addr2

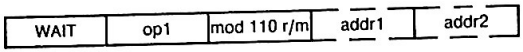
8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 m2rm	CD 19 m2rm	87+EA (84-90)+EA	mem-op ← ST (short-real)
9B D0 m2rm	CD 1D m2rm	100+EA (96-104)+EA	mem-op ← ST (long-real)

FSTCW
FNSTCW = Store Control Word

WAIT	op1	mod 111 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 m7rm	CD 19 m7rm	15+EA (12-18)+EA	mem-op ← processor control word
90 D9 m7rm	CD 19 m7rm	15+EA (12-18)+EA	mem-op ← processor control word (no wait)

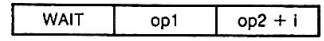
FSTENV
FNSTENV = Store Environment



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 m6rm	CD 19 m6rm	45+EA (40-50)+EA	mem-op ← 8087 environment
90 D9 r6rm	CD 19 m6rm	45+EA (40-50)+EA	mem-op ← 8087 environment (no wait)

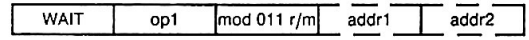
FSTP = Store Real and Pop

Stack top to Stack element



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DD D8+I	CD 1D D8+i	20 17-24	ST(i) ← ST pop stack

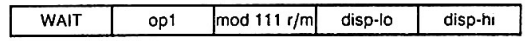
Stack top to memory operand



Long Real or Short Real

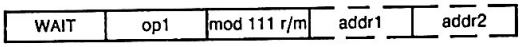
8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 m3rm	CD 19 m3rm	89+EA (86-92)+EA	mem-op ← ST pop stack (short-real)
9B DB m3rm	CD 1B m3rm	102+EA (98-106)+EA	mem-op ← ST pop stack (long-real)

Temp Real



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DD m7rm	CD 1D m7rm	55+EA (52-58)+EA	mem-op ← ST pop stack (temp-real)

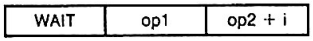
FSTSW
FNSTSW = Store Status Word



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DD m7rm	CD 1D m7rm	15+EA (12-18)+EA	mem-op←8087 status word
90 DD m7rm	CD 1D m7rm	15+EA (12-18)+EA	mem-op←8087 status word (no wait)

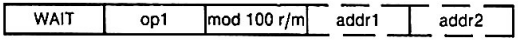
FSUB = Subtract Real

Stack top and Stack element



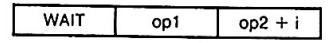
8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 E0+i	CD 18 E0+i	85 70-100	ST←ST — ST(i)
9B DC E8+i	CD 1C E8+i	85 70-100	ST(i)←ST(i) — ST

Stack top and memory operand



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 m4rm	CD 18 m4rm	105+EA (90-120)+EA	ST←ST — mem-op (short-real)
9B DC m4rm	CD 1C m4rm	110+EA (95-125)+EA	ST←ST — mem-op (long-real)

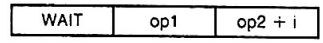
FSUBP = Subtract Real and Pop



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DE E9	CD 1E E9	90 75-105	ST(1)←ST(1) — ST pop stack
9B DE E8+i	CD 1E E8+i	90 75-105	ST(i)←ST(i) — ST pop stack

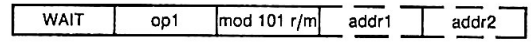
FSUBR = Subtract Real Reversed

Stack top and Stack element



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 E8+i	CD D8 E8+i	87 70-100	ST←ST(i) — ST
9B DC E0+i	CD 1C E0+i	87 70-100	ST(i)←ST — ST(i)

Stack top and memory operand



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 m5rm	CD 18 m5rm	105+EA (90-120)+EA	ST←mem-op — ST (short-real)
9B DC m5rm	CD 1C m5rm	110+EA (95-125)+EA	ST←mem-op — ST (long-real)

FSUBRP = Subtract Real Reversed and Pop

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DE E1	CD 1E E1	90 75-105	ST(1) ← ST — ST(1) pop stack
9B DE E0+i	CD 1E E0+i	90 75-105	ST(i) ← ST — ST(i) pop stack

FTST = Test Stack Top Against + 0.0

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 E4	CD 19 E4	42 38-48	ST ← ST — 0.0

FWAIT = (CPU) Wait While 8087 Is Busy

WAIT

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B	90	3+5n 3+5n	8086 wait instruction

FXAM = Examine Stack Top

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 E5	CD 19 E5	17 12-23	set condition code

FXCH = Exchange Registers

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 C8	CD 19 C8	12 10-15	$T_1 \leftarrow ST(1)$ ST(1) ← ST ST ← T_1
9B D9 C8+i	CD 19 C8+i	12 10-15	$T_1 \leftarrow ST(i)$ ST(i) ← ST ST ← T_1

FXTRACT = Extract Exponent and Significand

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F4	CD 19 F4	50 27-55	$T_1 \leftarrow$ exponent (ST) $T_2 \leftarrow$ significand (ST) ST ← T_1 push stack ST ← T_2

FYL2X = Compute Y * Log_2 X

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F1	CD 19 F1	950 900-1100	$T_1 \leftarrow ST(1) * \log_2 (ST)$ pop stack $ST \leftarrow T_1$

FYL2XP1 = Compute Y * Log_2 (X+1)

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F9	CD 19 F9	850 700-1000	$T_1 \leftarrow ST + 1$ $T_2 \leftarrow ST(1) * \log_2 T_1$ pop stack $ST \leftarrow T_2$

Assembler Controls Summary

Default control shown in italics

PRIMARY CONTROLS

Control	Effect
DATE(d)	System Date
DEBUG/*NODEBUG* DA DB/*NODB*	DEBUG puts local symbols information into object file for debugging. NODEBUG suppresses loading of local symbols information.
ERRORPRINT/*NOERRORPRINT* EP/*NOEP*	ERRORPRINT creates a file containing a listing of source line errors. NOERRORPRINT suppresses creation of that file.
MACRO/NOMACRO *MR*/NOMR	MACRO specifies that macro processor language will be recognized in source files. NOMACRO specifies nonrecognition of macros. They are scanned as is normal assembly language.
MOD186/*8086 mode* M1	MOD186 specifies that the iAPX 186 instruction set be recognized. The default is 8086 instructions only.
OBJECT/NOOBJECT *OJ*NOOJ	OBJECT specifies the creation of tan object module in the file specified. NOOBJECT specifies that an object module is not to be created.
PAGELENGTH(n) PL(n)	Specifies number (n) of printed lines per page in print file. Minimum pagelength is 20. Default is 60 lines per page.
PAGEWIDTH (n) PW (n)	Specifies the number (n) of characters, or columns, per line in the print and the errorprint files. Minimum is 60, maximum is 255. Default is 120.
PAGING/NOPAGING *PI*/NOPI	PAGING specifies that print file is to be formatted into pages with header at top of each page. NOPAGING specifies no formatting into pages.

PRINT/NOPRINT *PR*/NOPR	PRINT specifies that a source listing will be created during assembly. If no filename is specified, the source listing is written to the source file with the extension .LST appended. NOPRINT specifies that no source listing will be created.
SYMBOLS/*NOSYMBOLS* SB/*NOSB*	SYMBOLS specifies that a symbol listing table will be appended to the source listing in print file. NOSYMBOLS suppresses symbol table listing.
TYPE/*NOTYPE* TY/*NOTY*	TYPE specifies that type information be put into the object module. NOTYPE specifies that no type information be put into the object module.
WORKFILES WF	WORKFILES specifies the devices or directories used for storage of assembler-created temporary workfiles.
XREF/*NOXREF* XR/*NOXR*	XREF specifies that a symbol table, including line numbers, be appended to the source listing in print file. NOXREF specifies that no cross-reference line numbers are to be included.

GENERAL CONTROLS

EJECT EJ	Next line of source listing to be placed on new page.
GEN/*GENONLY*/NOGEN GE/*GO*/NOGE	Specify mode of listing assembler source text, macro calls and macro text in print file. GEN produces a listing that includes all source text, macro calls and expansion of each macro. GENONLY produces a listing that includes only source file non-macro text, and final result text for each macro called. NOGEN produces a listing that includes only the source file text.
INCLUDE IC	Causes subsequent source lines to be input from specified file.

LIST/NOLIST *LI*/NOLI	LIST specifies that listing of source program in print file is to resume with next source line read. NOLIST specifies that listing of source program in print file, beginning with next source line, is to be suppressed.
SAVE/RESTORE SA/RS	SAVE specifies that current setting of general controls be saved on a stack. RESTORE specifies that general controls be set to values stored on stack.
TITLE TT	Specifies the character string to appear on page header. Default title is module name specified in assembler NAME directive.

ASM86 Invocation under DOS

The following are instructions for invoking ASM86 on an IBM PC AT or IBM PC XT. Version 3.0 or greater of DOS is required. When the DOS prompt is displayed, you can invoke ASM86 as follows:

C>ASM86 *sourcepath* [*controls*] <cr>

where

ASM86	is the name of the ASM86 Macro Assembler.
sourcepath	is the pathname of the file containing the assembly language source module.
controls	is an optional sequence of assembler controls (and their parameters, if any) as defined in Chapter 3 of the *ASM86 Macro Assembler Operating Instructions for DOS Systems.*

DOS places a 128 character limit on the length of each line in the invocation sequence. To continue an invocation line, enter an ampersand (&) at the end of the line and press a carriage return. Two right angle brackets (>>) will appear on the next line. This is a prompt for you to continue the invocation line.

Assembler Directives

Symbol Definition:

EQU
LABEL
PURGE

Memory Reservation and Data Definition:

DB
DW
DD
DQ
DT
RECORD

Location Counter and Segmentation Control:

SEGMENT/ENDS
ORG
GROUP
ASSUME
PROC/ENDP
CODEMACRO/ENDM

Program Linkage:

NAME
PUBLIC
EXTRN
END

Processor Reset Register Initialization

Flags = 0000H (to disable interrupts and single-stepping)

CS = FFFFH
IP = 0000H (to begin execution at FFFF0H)

DS = 0000H
SS = 0000H
ES = 0000H

No other registers are acted upon during reset.

MCS®-86 Reserved Locations

Reserved Memory Locations

Intel Corporation reserves the use of memory location FFFF0H through FFFFFH (with the exception of FFFF0H - FFFF5H for JMP instr.) for Intel hardware and software products. If you use these locations for some other purpose, you may preclude compatibility of your system with certain of these products.

Reserved Input/Output Locations

Intel Corporation reserves the use of input/output locations F8H through FFH for Intel hardware and software products. Users who wish to maintain compatibility with present and future Intel products should not use these locations.

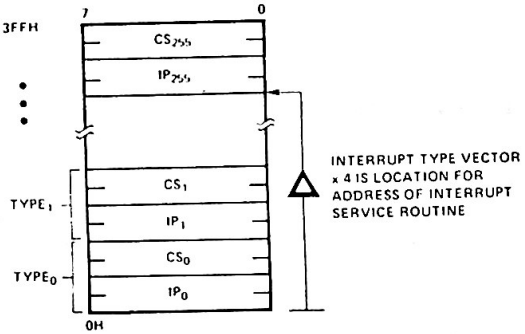
Reserved Interrupt Locations

Intel Corporation reserves the use of interrupts 0-31 (locations 00H through 7FH) for Intel hardware and software products. Users who wish to maintain compatibility with present and future Intel products should not use these locations.

Interrupts 0 through 4 (00H-13H) currently have dedicated hardware functions as defined below.

Interrupt	Location	Function
0	00H-03H	Divide by zero
1	04H-07H	Single step
2	08H-0BH	Non-maskable interrupt
3	0CH-0FH	One-byte interrupt instruction
4	10H-13H	Interrupt on overflow

Interrupt Pointer Table



iAPX 86/88/186 Instruction Set Matrix

Hi \ Lo	0	1	2	3	4	5	6	7
0	ADD b,f,r/m	ADD w,f,r/m	ADD b,f,r/m	ADD w,f,r/m	ADD b,ia	ADD w,ia	PUSH ES	POP ES
1	ADC b,f,r/m	ADC w,f,r/m	ADC b,f,r/m	ADC w,f,r/m	ADC b,i	ADC w,i	PUSH SS	POP SS
2	AND b,f,r/m	AND w,f,r/m	AND b,f,r/m	AND w,f,r/m	AND b,i	AND w,i	SEG ES	DAA
3	XOR b,f,r/m	XOR w,f,r/m	XOR b,f,r/m	XOR w,f,r/m	XOR b,i	XOR w,i	SEG SS	AAA
4	INC AX	INC CX	INC DX	INC BX	INC SP	INC BP	INC SI	INC DI
5	PUSH AX	PUSH CX	PUSH DX	PUSH BX	PUSH SP	PUSH BP	PUSH SI	PUSH DI
6	PUSHA	POPA	BOUND r,r/m					
7	JO	JNO	JB/ JNAE	JNB/ JAE	JE/ JZ	JNE/ JNZ	JBE/ JNA	JNBE/ JA
8	Immed b,r/m	Immed w,r/m	Immed b,r/m	Immed is,r/m	TEST b,r/m	TEST w,r/m	XCHG b,r/m	XCHG w,r/m
9	NOP	XCHG CX	XCHG DX	XCHG BX	XCHG SP	XCHG BP	XCHG SI	XCHG DI
A	MOV m→AL	MOV m→AX	MOV AL→m	MOV AX→m	MOVS b	MOVS w	CMPS b	CMPS w
B	MOV i→AL	MOV i→CL	MOV i→DL	MOV i→BL	MOV i→AH	MOV i→CH	MOV i→DH	MOV i→BH
C	Shift b,r/m,i	Shift w,r/m,i	RET (i-SP)	RET	LES	LDS	MOV b,r/m	MOV w,r/m
D	Shift b	Shift w	Shift b,v	Shift w,v	AAM	AAD		XLAT
E	LOOPNZ/ LOOPNE	LOOPZ/ LOOPE	LOOP	JCXZ	IN b	IN w	OUT b	OUT w
F	LOCK		REP	REP Z	HLT	CMC	Grp1 b,r/m	Grp1 w,r/m

where

mod r/m	000	001	010	011	100	101	110	111
Immed	ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
Shift	ROL	ROR	RCL	RCR	SHL/SAL	SHR	SHL/SAL	SAR
Grp1	TEST	–	NOT	NEG	MUL	IMUL	DIV	IDIV
Grp2	INC	DEC	CALL id	CALL lid	JMP id	JMP lid	PUSH	–

– 186 only instruction

iAPX 86/88/186 Instruction Set Matrix (Cont'd.)

Hi \ Lo	8	9	A	B	C	D	E	F
0	OR b,f,r/m	OR w,f,r/m	OR b,f,r/m	OR w,f,r/m	OR b,i	OR w,i	PUSH CS	/
1	SBB b,f,r/m	SBB w,f,r/m	SBB b,f,r/m	SBB w,f,r/m	SBB b,i	SBB w,i	PUSH DS	POP DS
2	SUB b,f,r/m	SUB w,f,r/m	SUB b,f,r/m	SUB w,f,r/m	SUB b,i	SUB w,i	SEG CS	DAS
3	CMP b,f,r/m	CMP w,f,r/m	CMP b,f,r/m	CMP w,f,r/m	CMP b,i	CMP w,i	SEG DS	AAS
4	DEC AX	DEC CX	DEC DX	DEC BX	DEC SP	DEC BP	DEC SI	DEC DI
5	POP AX	POP CX	POP DX	POP BX	POP SP	POP BP	POP SI	POP DI
6	PUSH iw	IMUL r,w,r/m	PUSH is	IMUL r,is,r/m	INS b	INS w	OUTS b	OUTS w
7	JS	JNS	JP/ JPE	JNP/ JPO	JL/ JNGE	JNL/ JGE	JLE/ JNG	JNLE/ JG
8	MOV b,f,r/m	MOV w,f,r/m	MOV b,f,r/m	MOV w,f,r/m	MOV sr,f,r/m	LEA	MOV sr,f,r/m	POP r/m
9	CBW	CWD	CALL i,d	WAIT	PUSHF	POPF	SAHF	LAHF
A	TEST b,i	TEST w,i	STOS b	STOS w	LODS b	LODS w	SCAS b	SCAS w
B	MOV i → AX	MOV i → CX	MOV i → DX	MOV i → BX	MOV i → SP	MOV i → BP	MOV i → SI	MOV i → DI
C	ENTER iw,ib	LEAVE	RET f,(i-SP)	RET f	INT Type 3	INT (Any)	INTO	IRET
D	ESC 0	ESC 1	ESC 2	ESC 3	ESC 4	ESC 5	ESC 6	ESC 7
E	CALL d	JMP d	JMP r,d	JMP si d	IN v,b	IN v,w	OUT v,d	OUT v,w
F	CLC	STC	CLI	STI	CLD	STD	Grp 2 b,r/m	Grp 2 w,r/m

b = byte operation
d = direct
f = from CPU reg
i = immediate
ia = immed. to accum.
ib = immediate byte
id = indirect
is = immed. byte sign ext.
iw = immediate word
l = long ie intersegment
m = memory
r = register
r/m = EA is second byte
si = short intrasegment
sr = segment register
t = to CPU reg
v = variable
w = word operation
z = zero

Clocks for MOD186 Operation

FUNCTION	FORMAT				186 Clock Cycles
DATA TRANSFER					
MOV = Move					
Register to Register/Memory	1 0 0 0 1 0 0 w	mod reg r/m			2/12
Register memory to register	1 0 0 0 1 0 1 w	mod reg r/m			2/9
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1	12-13
Immediate to register	1 0 1 1 w reg	data	data if w = 1		3-4
Memory to accumulator	1 0 1 0 0 0 0 w	addr low	addr high		9
Accumulator to memory	1 0 1 0 0 0 1 w	addr low	addr high		8
Register/memory to segment register	1 0 0 0 1 1 1 0	mod sreg r/m			2/9
Segment register to register/memory	1 0 0 0 1 1 0 0	mod sreg r/m			2/11
PUSH = Push					
Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m			16
Register	0 1 0 1 0 reg				10
Segment register	0 0 0 reg 1 1 0				9
Immediate	0 1 1 0 1 0 s 0	data	data if s = 0		10
PUSHA = Push All	0 1 1 0 0 0 0 0				36
POP = Pop					
Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m			20
Register	0 1 0 1 1 reg				10
Segment register	0 0 0 reg 1 1 1	(reg ≠ 01)			8
POPA = Pop All	0 1 1 0 0 0 0 1				51
XCHG = Exchange					
Register/memory with register	1 0 0 0 0 1 1 w	mod reg r/m			4/17
Register with accumulator	1 0 0 1 0 reg				3
IN = Input from					
Fixed port	1 1 1 0 0 1 0 w	port			10
Variable port	1 1 1 0 1 1 0 w				8
OUT = Output to					
Fixed port	1 1 1 0 0 1 1 w	port			9
Variable port	1 1 1 0 1 1 1 w				7
XLAT Translate byte to AL	1 1 0 1 0 1 1 1				11
LEA Load EA to register	1 0 0 0 1 1 0 1	mod reg r/m			6
LDS Load pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m		(mod ≠ 11)	18
LES Load pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m		(mod ≠ 11)	18
LAHF Load AH with flags	1 0 0 1 1 1 1 1				2
SAHF Store AH into flags	1 0 0 1 1 1 1 0				3
PUSHF Push flags	1 0 0 1 1 1 0 0				9
POPF Pop flags	1 0 0 1 1 1 0 1				8

Clocks for MOD186 Operation (Cont'd.)

FUNCTION	FORMAT	186 Clock Cycles
ARITHMETIC		
ADD = Add		
Reg memory with register to either	000000dw mod reg r/m	3/10
Immediate to register memory	100000sw mod000 r/m data data if s:w = 01	4/16
Immediate to accumulator	0000010w data data if w = 1	3/4
ADC = Add with carry		
Reg memory with register to either	000100dw mod reg r/m	3/10
Immediate to register memory	100000sw mod010 r/m data data if s:w = 01	4/16
Immediate to accumulator	0001010w data data if w = 1	3/4
INC = Increment		
Register memory	1111111w mod000 r/m	3/15
Register	01000 reg	3
SUB = Subtract		
Reg memory and register to either	001010dw mod reg r/m	3/10
Immediate from register memory	100000sw mod101 r/m data data if s:w = 01	4/16
Immediate from accumulator	0010110w data data if w = 1	3/4
SBB = Subtract with borrow		
Reg memory and register to either	000110dw mod reg r/m	3/10
Immediate from register memory	100000sw mod011 r/m data data if s:w = 01	4/16
Immediate from accumulator	0001110w data data if w = 1	3/4
DEC = Decrement		
Register memory	1111111w mod001 r/m	3/15
Register	01001 reg	3
CMP = Compare		
Register memory with register	0011101w mod reg r/m	3/10
Register with register memory	0011100w mod reg r/m	3/10
Immediate with register memory	100000sw mod111 r/m data data if s:w = 01	3/10
Immediate with accumulator	0011110w data data if w = 1	3/4
NEG = Change sign	1111011w mod011 r/m	3
AAA = ASCII adjust for add	00110111	8
DAA = Decimal adjust for add	00100111	4
AAS = ASCII adjust for subtract	00111111	7
DAS = Decimal adjust for subtract	00101111	4
MUL = Multiply (unsigned)	1111011w mod100 r/m	
Register Byte		26–28
Register Word		35–37
Memory Byte		32–34
Memory Word		41–43
IMUL = Integer multiply (signed)	1111011w mod101 r/m	
Register Byte		25–28
Register Word		34–37
Memory Byte		31–34
Memory Word		40–43
IMUL = Integer immediate multiply (signed)	011010s1 mod reg r/m data data if s = 0	22–25/29–32
DIV = Divide (unsigned)	1111011w mod110 r/m	
Register Byte		29
Register Word		38
Memory Byte		35
Memory Word		44

Clocks for MOD186 Operation (Cont'd.)

FUNCTION	FORMAT	186 Clock Cycles
ARITHMETIC (Continued)		
IDIV = Integer divide (signed)	1111011w mod111 r/m	
Register Byte		44–52
Register Word		
Memory Byte		53–61
Memory Word		50–58
AAM = ASCII adjust for multiply	11010100 00001010	59–67
AAD = ASCII adjust for divide	11010101 00001010	19
CBW = Convert byte to word	10011000	15
CWD = Convert word to double word	10011001	2
		4
LOGIC		
Shift, Rotate Instructions		
Register Memory by 1	110100vw mod TTT r/m	2/15
Register Memory by CL	110100vw mod TTT r/m	5 + n/17 + n
Register Memory by Count	1100000w mod TTT r/m count	5 + n/17 + n

TTT	Instruction
000	ROL
001	ROR
010	RCL
011	RCR
100	SHL/SAL
101	SHR
111	SAR

AND = And		
Reg memory and register to either	001000dw mod reg r/m	3/10
Immediate to register memory	1000000w mod100 r/m data data if w = 1	4/16
Immediate to accumulator	0010010w data data if w = 1	3/4
TEST = And function to flags, no result		
Register memory and register	1000010w mod reg r/m	3/10
Immediate data and register memory	1111011w mod000 r/m data data if w = 1	4/10
Immediate data and accumulator	1010100w data data if w = 1	3/4
OR = Or		
Reg memory and register to either	000010dw mod reg r/m	3/10
Immediate to register memory	1000000w mod001 r/m data data if w = 1	4/16
Immediate to accumulator	0000110w data data if w = 1	3/4
XOR = Exclusive or		
Reg memory and register to either	001100dw mod reg r/m	3/10
Immediate to register memory	1000000w mod110 r/m data data if w = 1	4/16
Immediate to accumulator	0011010w data data if w = 1	3/4
NOT = Invert register memory	1111011w mod010 r/m	3
STRING MANIPULATION		
MOVS = Move byte word	1010010w	14
CMPS = Compare byte word	1010011w	22
SCAS = Scan byte word	1010111w	15
LODS = Load byte wd to AL AX	1010110w	12
STOS = Stor byte wd from AL A	1010101w	10
INS = Input byte/wd from DX port	0110110w	14
OUTS = Output byte/wd to DX port	0110111w	14

Clocks for MOD186 Operation (Cont'd.)

FUNCTION	FORMAT	186 Clock Cycles
STRING MANIPULATION (Continued)		
Repeated by count in CX		
MOVS Move string	1 1 1 1 0 0 1 0 | 1 0 1 0 0 1 0 w	8+8n
CMPS Compare string	1 1 1 1 0 0 1 z | 1 0 1 0 0 1 1 w	5+22n
SCAS Scan string	1 1 1 1 0 0 1 z | 1 0 1 0 1 1 1 w	5+15n
LODS Load string	1 1 1 1 0 0 1 0 | 1 0 1 0 1 1 0 w	6+11n
STOS Store string	1 1 1 1 0 0 1 0 | 1 0 1 0 1 0 1 w	6+9n
INS = Input string	1 1 1 1 0 0 1 0 | 0 1 1 0 1 1 0 w	8+8n
OUTS = Output string	1 1 1 1 0 0 1 0 | 0 1 1 0 1 1 1 w	8+8n
CONTROL TRANSFER		
CALL = Call		
Direct within segment	1 1 1 0 1 0 0 0 | disp-low | disp-high	14
Register memory indirect within segment	1 1 1 1 1 1 1 1 | mod 0 1 0 r/m	13/19
Direct intersegment	1 0 0 1 1 0 1 0 | segment offset | segment selector	23
Indirect intersegment	1 1 1 1 1 1 1 1 | mod 0 1 1 r/m | (mod ≠ 11)	38
JMP = Unconditional jump		
Short/long	1 1 1 0 1 0 1 1 | disp-low	13
Direct within segment	1 1 1 0 1 0 0 1 | disp-low | disp-high	13
Register memory indirect within segment	1 1 1 1 1 1 1 1 | mod 1 0 0 r/m	11/17
Direct intersegment	1 1 1 0 1 0 1 0 | segment offset | segment selector	13
Indirect intersegment	1 1 1 1 1 1 1 1 | mod 1 0 1 r/m | (mod ≠ 11)	26
RET = Return from CALL		
Within segment	1 1 0 0 0 0 1 1	16
Within seg adding immed to SP	1 1 0 0 0 0 1 0 | data-low | data-high	18
Intersegment	1 1 0 0 1 0 1 1	22
Intersegment adding immed to SP	1 1 0 0 1 0 1 0 | data-low | data-high	25

Clocks for MOD186 Operation (Cont'd.)

FUNCTION	FORMAT	186 Clock Cycles
CONTROL TRANSFER (Continued)		
JE JZ Jump if equal/zero	0 1 1 1 0 1 0 0 | disp	4/13
JL JNGE Jump if less/not greater or equal	0 1 1 1 1 1 0 0 | disp	4/13
JLE JNG Jump if less or equal/not greater	0 1 1 1 1 1 1 0 | disp	4/13
JB JNAE Jump if below/not above or equal	0 1 1 1 0 0 1 0 | disp	4/13
JBE JNA Jump if below or equal/not above	0 1 1 1 0 1 1 0 | disp	4/13
JP JPE Jump if parity/parity even	0 1 1 1 1 0 1 0 | disp	4/13
JO Jump if overflow	0 1 1 1 0 0 0 0 | disp	4/13
JS Jump if sign	0 1 1 1 1 0 0 0 | disp	4/13
JNE JNZ Jump if not equal/not zero	0 1 1 1 0 1 0 1 | disp	4/13
JNL JGE Jump if not less/greater or equal	0 1 1 1 1 1 0 1 | disp	4/13
JNLE JG Jump if not less or equal/greater	0 1 1 1 1 1 1 1 | disp	4/13
JNB JAE Jump if not below/above or equal	0 1 1 1 0 0 1 1 | disp	4/13
JNBE JA Jump if not below or equal/above	0 1 1 1 0 1 1 1 | disp	4/13
JNP JPO Jump if not parity/parity odd	0 1 1 1 1 0 1 1 | disp	4/13
JNO Jump if not overflow	0 1 1 1 0 0 0 1 | disp	4/13
JNS Jump if not sign	0 1 1 1 1 0 0 1 | disp	4/13
LOOP Loop CX times	1 1 1 0 0 0 1 0 | disp	5/15
LOOPZ LOOPE Loop while zero/equal	1 1 1 0 0 0 0 1 | disp	6/16
LOOPNZ LOOPNE Loop while not zero/equal	1 1 1 0 0 0 0 0 | disp	6/16
JCXZ Jump on CX zero	1 1 1 0 0 0 1 1 | disp	6/16
ENTER = Enter Procedure	1 1 0 0 1 0 0 0 | data-low | data-high | L	5
L = 0		15
L = 1		25
L > 1		22+16(n−1)
LEAVE = Leave Procedure	1 1 0 0 1 0 0 1	8
INT = Interrupt		
Type specified	1 1 0 0 1 1 0 1 | type	47
Type 3	1 1 0 0 1 1 0 0	45
INTO = Interrupt on overflow	1 1 0 0 1 1 1 0	48/4
IRET = Interrupt return	1 1 0 0 1 1 1 1	28
BOUND = Detect value out of range	0 1 1 0 0 0 1 0 | mod reg r/m	33–35

intel

3065 Bowers Avenue, Santa Clara, California 95052-8130
(408) 987-8080

Printed in U S.A.

SOFTWARE